System programming (03KTOOT)

Laboratory #2: **Processes**

Laboratories are mandatory. Each student must submit the work (exclusively in .zip format, with text in .pdf and code (if required) in a format able to be run on the LABINF computers) following the rules published on the web portal. Please remember that for each laboratory the final mark can get from -2.0 (not submitted or with severe flaws) up to +0.5 points (excellent work). If all 4 laboratories are not submitted 8 points are lost.

The deadline for the second laboratory is Nov 20nd 2015 at 7 p.m.

Exercise 1: Write a C program able to read a list of integer numbers from a text file (whose name is given as a **first command line parameter**) printing on another text file (whose name is given as a **second command line parameter**) only the numbers that are prime. Please use the appropriate UNIX system calls for reading and writing files.

E.g. given the following command ling:

```
$> prime.out   first.dat   second.dat
```

and given the content of first.dat file:
4
6
7
3

the program will create a second file second.dat:
7
3

Exercise 2: Write a simplified version of the ls command receiving as a command line parameter the name of a directory and that visualizes the names of all the files contained in this directory

Exercise 3: Write a program able to copy data contained in a file whose name is a command line parameter in a second file whose name is passed as a second parameter

Exercise 4: Write a simple program able to do a fork() and then to print messages both from the father and the son processes. Finally modify the program in order to have the messages printed from the father only after the son is finished.

Exercise 5: Write a C program able to run the following Unix commands:

```
cp /etc/passwd  .
sort passwd –o mypasswd
cat mypasswd
```

Exercise 6: Create a function mydup2() able to give the same functionalities of the dup2(), but written using only the dup().

Exercise 7: Write a program that creates a pipe and after the fork() uses the pipe to send a large file from one process to the other during the execution.

Exercise 8: Write a simple producer-consumer program, in which the queue between the two processes (father and son, or two brothers) is at most of 4 tokens, each token being a character. Use semaphores to protect the producer-consumer operations.

Exercise 9: Write a C program that performs the following operations:

- Creates a child process that generates a file called random.txt containing a list of random numbers separated by the space character. Let the length of this list be a random value too.
- When the file is created the program must instantiate three sub-processes:
  - The first process must execute the linux command wc –m to print on the screen the number of bytes of the file.
  - The second process must execute the linux command wc –w to count the number of words in the file
  - The third process must execute a custom program designed to compute the average of all generated number.
- The main process terminates when all three children terminate.

Exercise 10: Create a modified Linux shell called my shell. The new shell must have the following characteristic:

- It receives commands on the command line like the standard linux shell (the command cannot be split on different lines).
- Every command is executed in background in a dedicated child process
- The output of the command is not written on the screen but is saved in a file named <PID>.log where <PID> is the process identifier of the child process executing the command.
- The shell terminates when the user enters the exit command.

Exercise 11: Write a multi-process program that evaluates the following math series:

$$\sum_0^N 2^i$$

The main process receives the N value as input (set max value for N to 5) and performs the final sum. Each 2^i is evaluated by the i-th process and sent to the father for the final sum.

Exercise 12: Use the previous program skeleton to evaluate the following Taylor expansion:

$$e^x = 1 + \sum_1^N \frac{x^i}{i!}$$

The main process gets x and N value as input (x is a double number and N > 0) and performs the final sum. No boundaries are set to N value.

Each $\frac{x^i}{i!}$ is performed by the i-th process and sent to the father for the final sum.